# vplyv synchrónnosti

## algoritmy založené na porovnaniach

- ekvivalentné okolia
- $c$-symetrický reťazec: pre každé $\sqrt{n} \leq l \leq n$ a pre každý segment $S$ je $\lfloor \frac{cn}{l} \rfloor$ ekvivalentných
- bit-reversal je $1/2$-symetrický
- $c$-symetrický reťazec, alg. nemôže skončiť po $k$ kolách pre $\left\lfloor \frac{cn}{2k+1} \right\rfloor \geq 2$
- počet správ: $k = \left\lfloor \frac{cn-2}{4} \right\rfloor$, je aspoň $k+1$ kôl
- aspoň $\left\lfloor \frac{cn}{2r-1} \right\rfloor$ aktívnych v $r$-tom kole pošle správu

## algoritmy využívajúce integer ID

- rôzne rýchlosti
- v $i$-tom kroku test

# cvičenie

V toruse rozmerov $n \times n$ (t.j. zacyklenej mriežke) sú na začiatku zobudené dva vrcholy. Napíšte algoritmus, pomocou ktorého sa každý z nich dozvie identifikátor druhého s použitím $O(n)$ správ.

Nájdite asymptoticky optimálny (čo do počtu správ) algoritmus na voľbu šéfa v úplnom bipartitnom grafe $K_{n,n}$. Dokážte jeho zložitosť a optimalitu.

**$\star$**

broadcasting a voľba šéfa na (ne?)orientovanej hyperkocke s lineárnym počtom správ

# routovanie

cieľ: doručovať srávy medzi ľubovoľnou dvojicou

## modely

- destination based
- splittable
- connections (wormhole)
- buffering
- selfish

## ciele

- statické váhy (najkratšie cesty)
- dynamické váhy (hot potato)
- deadlock

# najkratšie cesty

```
begin (* Initialize S to ∅ and D to ∅-distance *)
        S := ∅ ;
        forall u, v do
                if u = v then D[u, v] := 0
                else if uv ∈ E then D[u, v] := ω_uv
                else D[u, v] := ∞ ;
        (* Expand S by pivoting *)
        while S ≠ V do
           (* Loop invariant: ∀u, v : D[u, v] = d^S(u, v) *)
           begin pick w from V \ S ;
                   (* Execute a global w-pivot *)
                   forall u ∈ V do
                           (* Execute a local w-pivot at u *)
                           forall v ∈ V do
                                   D[u, v] := min ( D[u, v], D[u, w] + D[w, v] ) ;
                   S := S ∪ {w}
           end     (* ∀u, v : D[u, v] = d(u, v) *)
end
```

# najkratšie cesty

```
var S_u   : set of nodes ;
    D_u   : array of weights ;
    Nb_u  : array of nodes ;

begin S_u := ∅ ;
      forall v ∈ V do
            if v = u
              then begin D_u[v] := 0 ; Nb_u[v] := udef end
            else if v ∈ Neigh_u
              then begin D_u[v] := ω_uv ; Nb_u[v] := v end
            else begin D_u[v] := ∞ ; Nb_u[v] := udef end ;
      while S_u ≠ V do
        begin pick w from V \ S_u ;
              (* All nodes must pick the same node w here *)
              if u = w
                then "broadcast the table D_w"
                else "receive the table D_w"
              forall v ∈ V do
                    if D_u[w] + D_w[v] < D_u[v] then
                      begin D_u[v] := D_u[w] + D_w[v] ;
                            Nb_u[v] := Nb_u[w]
                      end ;
              S_u := S_u ∪ {w}
        end
end
```

# najkratšie cesty

```
var S_u   : set of nodes ;
    D_u   : array of weights ;
    Nb_u  : array of nodes ;

begin S_u := ∅ ;
      forall v ∈ V do
            if v = u
              then begin D_u[v] := 0 ; Nb_u[v] := udef end
            else if v ∈ Neigh_u
              then begin D_u[v] := ω_uv ; Nb_u[v] := v end
            else begin D_u[v] := ∞ ; Nb_u[v] := udef end ;
      while S_u ≠ V do
        begin pick w from V \ S_u ;
              (* Construct the tree T_w *)
              forall x ∈ Neigh_u do
                    if Nb_u[w] = x then send ⟨ ys, w ⟩ to x
                                   else send ⟨ nys, w ⟩ to x ;
              num_rec_u := 0 ; (* u must receive |Neigh_u| messages *)
              while num_rec_u < |Neigh_u| do
                    begin receive ⟨ ys, w ⟩ or ⟨ nys, w ⟩ message ;
                          num_rec_u := num_rec_u + 1
                    end ;
              if D_u[w] < ∞ then (* participate in pivot round *)
                begin if u ≠ w
                        then receive ⟨ dtab, w, D ⟩ fromthis Nb_u[w] ;
                      forall x ∈ Neigh_u do
                            if ⟨ ys, w ⟩ was received from x
                              then send ⟨ dtab, w, D ⟩ to x ;
                      forall v ∈ V do (* local w-pivot *)
                            if D_u[w] + D[v] < D_u[v] then
                              begin D_u[v] := D_u[w] + D[v] ;
                                    Nb_u[v] := Nb_u[w]
                              end
                end ;
```

# Netchange

```
var Neigh_u   : set of nodes ;   (* The neighbors of u *)
    D_u       : array of 0.. N ; (* D_u[v] estimates d(u, v) *)
    Nb_u      : array of nodes ; (* Nb_u[v] is preferred neighbor for v *)
    ndis_u    : array of 0.. N ; (* ndis_u[w, v] estimates d(w, v) *)

Initialization:
  begin forall w ∈ Neigh_u, v ∈ V do ndis_u[w, v] := N ;
        forall v ∈ V do
              begin D_u[v] := N ; Nb_u[v] := udef end ;
        D_u[u] := 0 ; Nb_u[u] := local ;
        forall w ∈ Neigh_u do send ⟨ mydist, u, 0 ⟩ to w
  end

Procedure Recompute (v):
  begin if v = u
          then begin D_u[v] := 0 ; Nb_u[v] := local end
          else begin (* Estimate distance to v *)
                     d := 1 + min{ndis_u[w, v] : w ∈ Neigh_u} ;
                     if d < N then
                       begin D_u[v] := d ;
                             Nb_u[v] := w with 1 + ndis_u[w, v] = d
                       end
                     else begin D_u[v] := N ; Nb_u[v] := udef end
               end ;
        if D_u[v] has changed then
          forall x ∈ Neigh_u do send ⟨ mydist, v, D_u[v] ⟩ to x
  end
```

```
Processing a ⟨ mydist, v, d ⟩ message from neighbor w:
    { A ⟨ mydist, v, d ⟩ is at the head of Q_wv }
    begin receive ⟨ mydist, v, d ⟩ from w ;
          ndis_u[w, v] := d ; Recompute (v)
    end

Upon failure of channel uw:
    begin receive ⟨ fail, w ⟩ ; Neigh_u := Neigh_u \ {w} ;
          forall v ∈ V do Recompute (v)
    end

Upon repair of channel uw:
    begin receive ⟨ repair, w ⟩ ; Neigh_u := Neigh_u ∪ {w} ;
          forall v ∈ V do
                begin ndis_u[w, v] := N ;
                      send ⟨ mydist, v, D_u[v] ⟩ to w
                end
    end
```

## korektnosť

lexikograficky klesá hodnota $[t_0, t_1, \ldots, t_N]$
kde $t_i$ je počet správ $\langle$**mydist**, $i\rangle$+ počet dvojíc $u, v$ kde $D_u[v] = i$